The Dual-Path Execution Model for Efficient GPU Control Flow

Minsoo Rhu, Mattan Erez
HPCA 2013

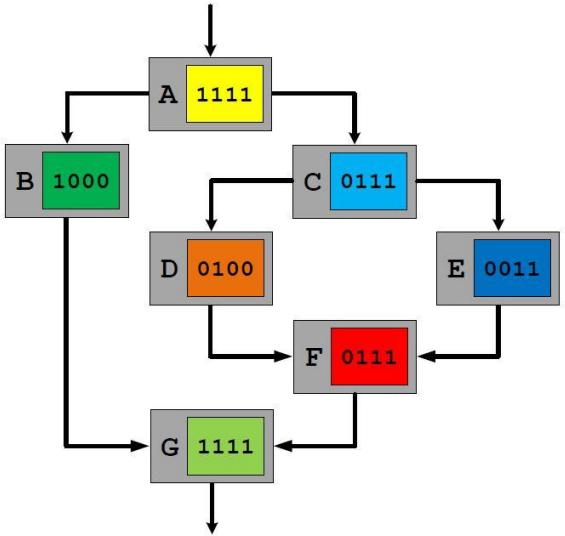
Presented by DJ Park, Romita Mullick, Hans Giesen

Outline

- Background
  - Stack-based reconvergence
  - Dynamic warp subdivision
- Dual-path execution model
- Evaluation
- Conclusion

Stack-Based Reconvergence

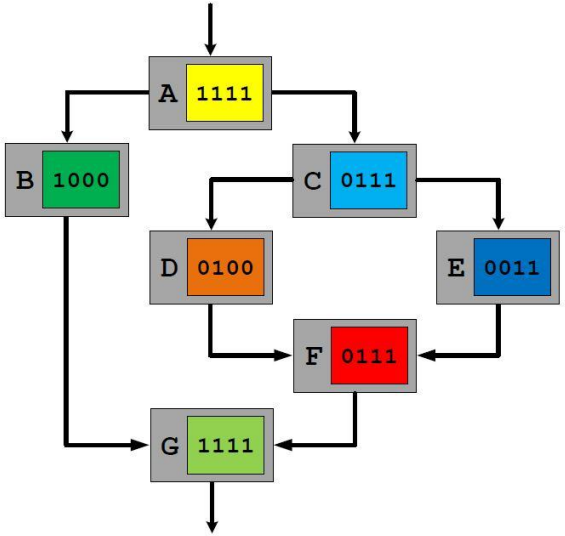
- When the control flow of different threads within a single warp diverges, execution of concurrent control paths is serialized with every divergence.
- Threads reconverge at the *immediate post-dominator*(PDOM) instruction of that branch



Stack-Based Reconvergence

- The way to implement reconvergence: treat control flow execution as a serial stack
- Each time control diverges, both the taken and not taken paths are pushed onto a stack (in arbitrary order) and the path at the new top of stack is executed
- When the control path reaches its reconvergence point, the entry is popped off of the stack and execution now follows the alternate direction of the diverging branch.

Reconvergence stack and its operation

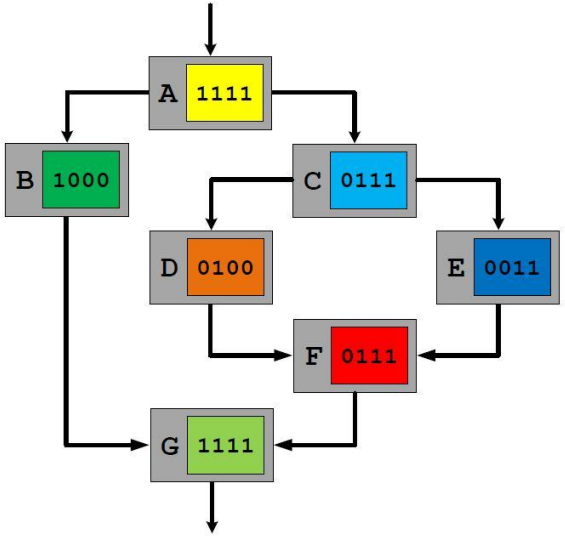


Single-path stack

	PC	Active Mask	RPC
TOS →	A	1111	-

(a) Initial status of the stack. The current TOS designates the fact that basic block A is being executed.

Reconvergence stack and its operation

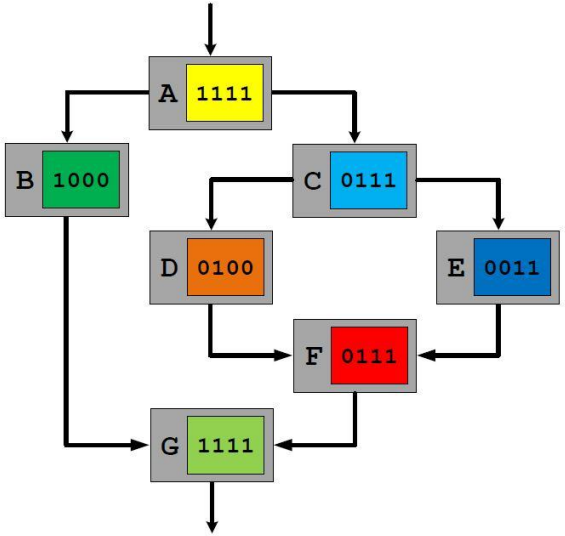


Single-path stack

	PC	Active Mask	RPC
	G	1111	-
	C	0111	G
TOS →	B	1000	G

(b) Two entries of block B and C are *pushed* into the stack when BR_{B-C} is executed. RPC is updated to block G.

Reconvergence stack and its operation

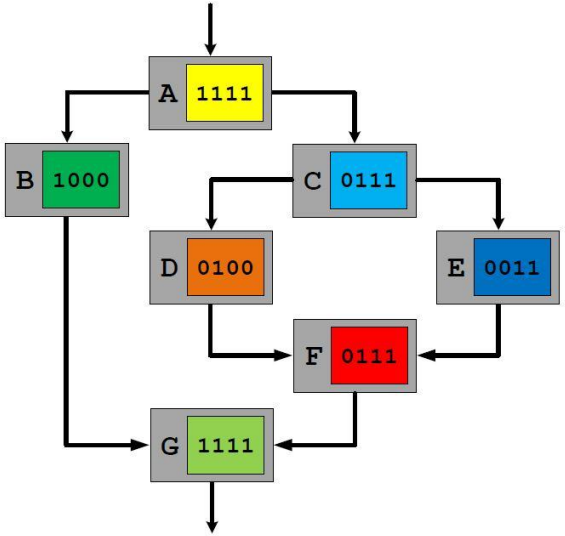


Single-path stack

	PC	Active Mask	RPC
	G	1111	-
TOS →	C	0111	G

(c) The stack entry, corresponding to block B at TOS, is *popped* out when PC matches RPC value of G.

Reconvergence stack and its operation

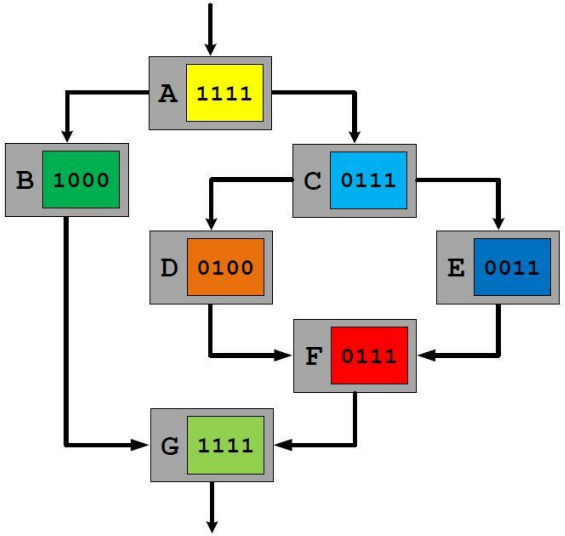


Single-path stack

	PC	Active Mask	RPC
	G	1111	-
	F	0111	G
	E	0011	F
TOS	D	0100	F

(d) Two more entries for block D and E are pushed into the stack when the warp executes $BR_{D\text{-}E}$.

Reconvergence stack and its operation

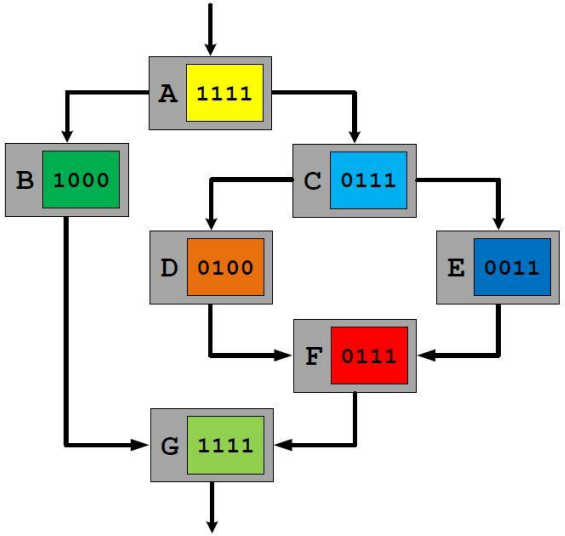


Single-path stack

	PC	Active Mask	RPC
	G	1111	-
TOS →	F	0111	G

(e) Threads are reconverged back at block F when both entries for block D and E are popped out.

Reconvergence stack and its operation

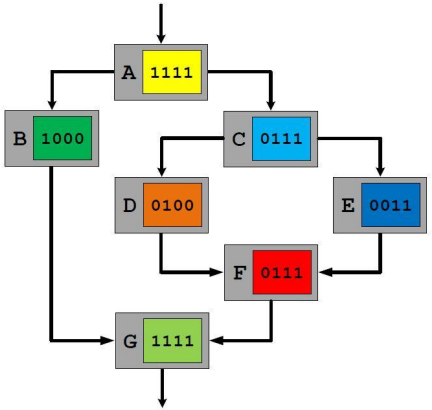


Single-path stack

	PC	Active Mask	RPC
TOS →	G	1111	-

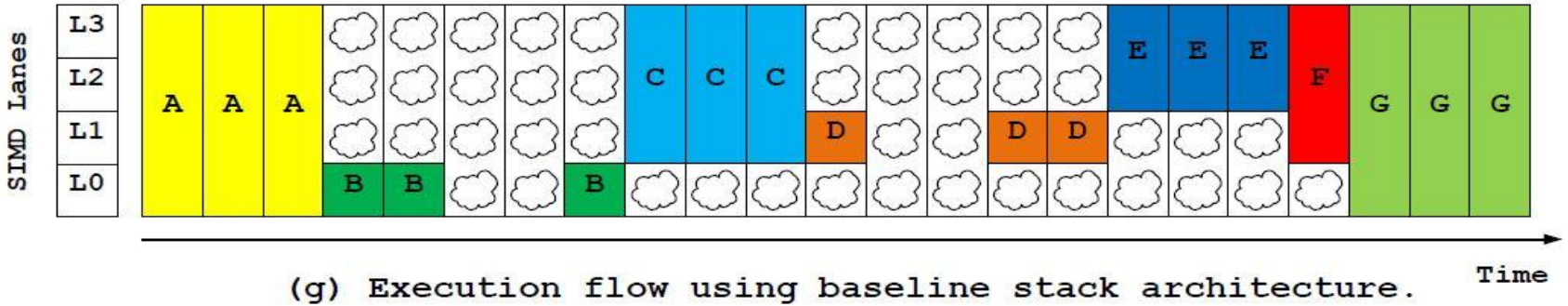
(f) All four threads become active again when the stack entry for block F is popped out.

Reconvergence stack and its operation



Deficiencies:

- SIMD utilization decreases every time control flow diverges
- Execution is serialized



(g) Execution flow using baseline stack architecture.

In Figure 2, are the idle slots in between block B the memory I/O time(cache-miss)?

Yes, cache-miss, long memory latency, etc

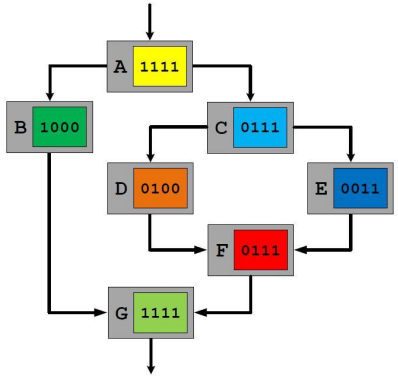
Dynamic Warp Subdivision

- Allow warps to interleave the scheduling of instructions from concurrently executable paths(left and right paths)

- A divergent branch may either utilize the baseline single-path stack, or instead, ignore the stack and utilize an additional hardware structure, the warp-split table (WST), which is used to track the independently-schedulable warp-splits

- Warp-split: independent scheduling entities and are treated equally as warps by the scheduler (the left and right paths of a divergent)

DWS operation



When $BR_{B\text{-}C}$ is executed, the warp is not subdivided because the number of instructions in block G(PDOM, and it has 3 insns) is larger than the subdivision threshold(which is 2 for this case).

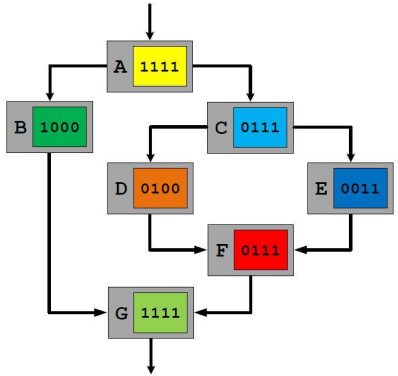
Single-path stack

	PC	Active Mask	RPC
	G	1111	-
	C	0111	G
TOS →	B	1000	G

Warp-split table

PC	Active Mask	RPC

DWS operation



$BR_{D\text{-}E}$ has a PDOM(F has 1 insn) smaller than the threshold(2) which allows the warp to be subdivided.

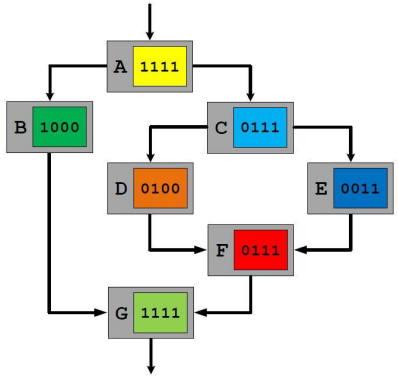
Single-path stack

	PC	Active Mask	RPC
	G	1111	-
TOS	C	0111	G

Warp-split table

PC	Active Mask	RPC
D	0100	G
E	0011	G

DWS operation



Note that RPC for two entries in warp-split table is G, not F

Single-path stack

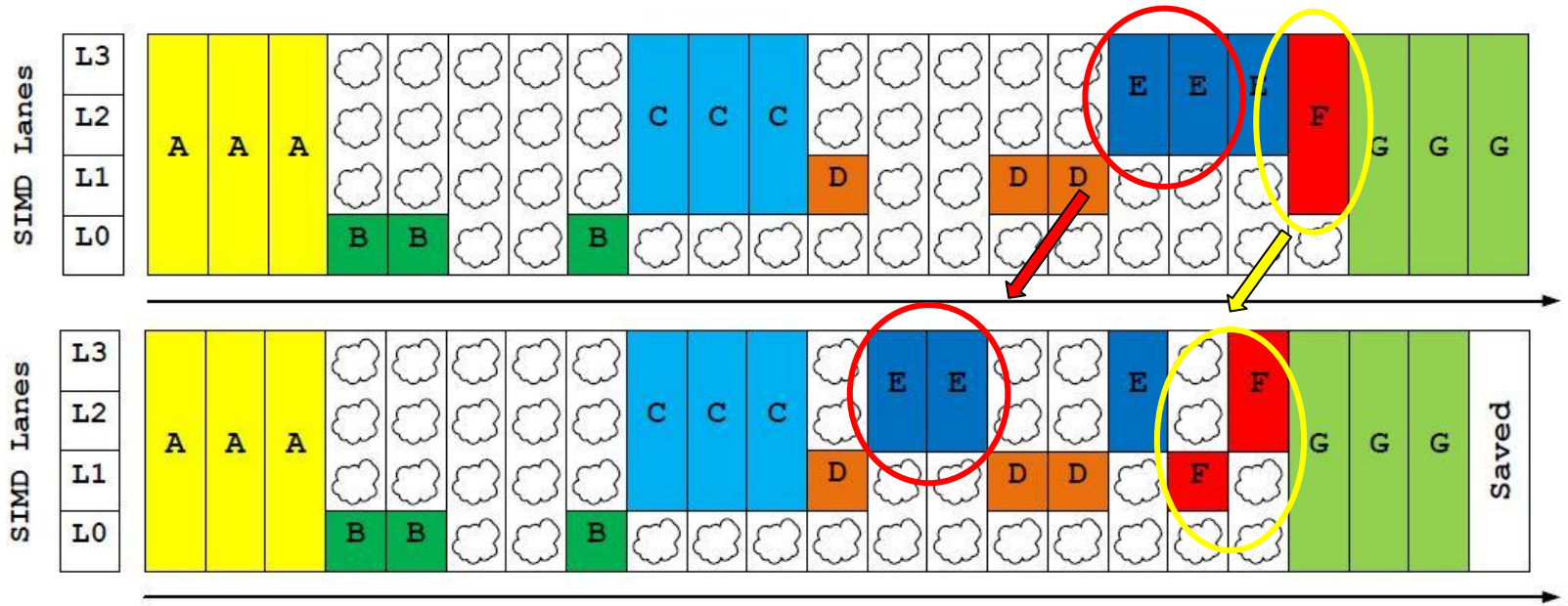
	PC	Active Mask	RPC
TOS	G	1111	-

Warp-split table

PC	Active Mask	RPC
F	0100	G
F	0011	G

DWS operation

Compared with baseline architecture: increases parallelism and potential latency hiding



Deficiency: reduced SIMD utilization (the stack could have reconverged nested branches whereas the WST cannot)

Comparing figure 3 & 4, I am a little confused here. In dual path method (figure 4), the three threads of block F is executed all at the same time. However, in DWS (figure 3), lane 1 was executed first. Could the presenter elaborate the comparison between DWS and dual path method?

Warp-splits continue executing asynchronously and keep being subdivided upon future divergent branches until they reach the PDOM associated with the top of the reconvergence stack

Motivation

- Single Path Execution maximizes SIMD utilization with structured control flow, but always serializes execution with only a single path schedulable at any given time

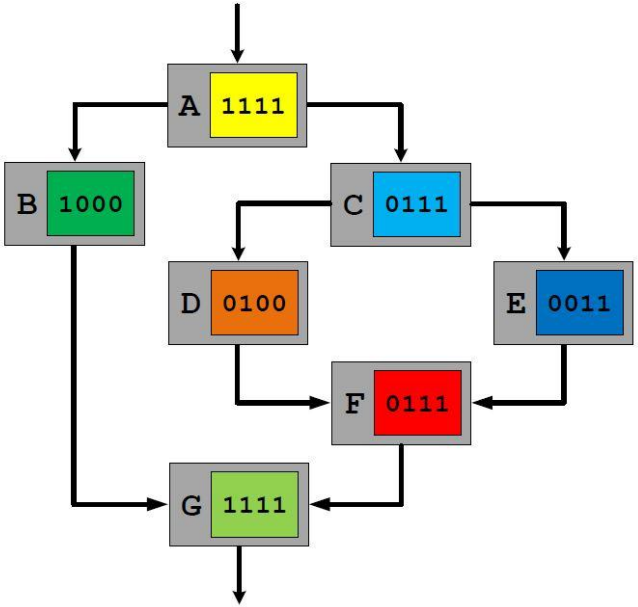
- Dynamic Warp Subdivision can interleave the scheduling of multiple paths and increase TLP, but this sacrifices SIMD lane utilization

- Goal: matches the utilization and SIMD efficiency of the baseline SPE while still enhancing TLP in some cases

Dual-Path execution model

- Dual-Path stack structure
  - Idea: instead of pushing the taken and fall-through paths onto the stack one after the other, in effect serializing their execution, the two paths are maintained in parallel.
  - Stack entry:
    - PC and active mask value of the left path (Path L)
    - PC and active mask value of the right path (Path R)
    - The RPC (reconvergence PC) of the two paths

Dual-Path execution

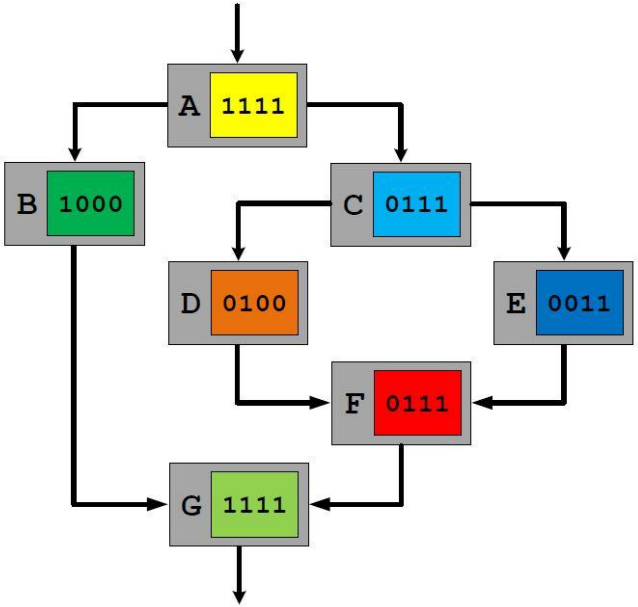


TOS

Dual-path stack

PC_L	$Mask_L$	PC_R	$Mask_R$	RPC
A	1111	-	-	-

Dual-Path execution

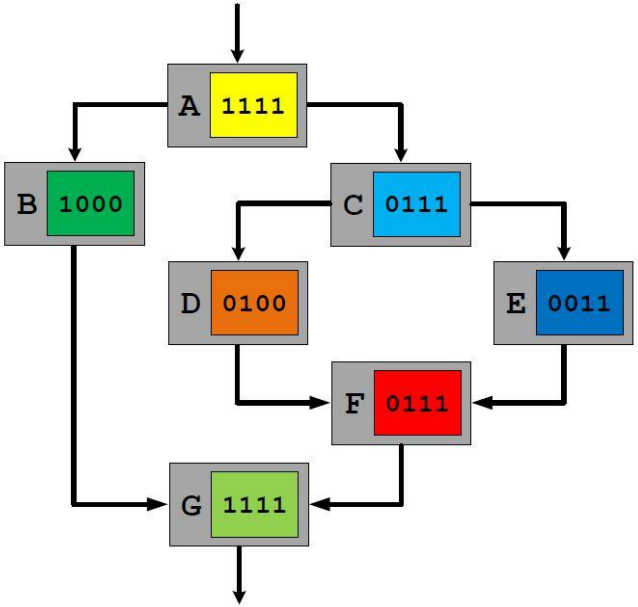


TOS

Dual-path stack

PC_L	$Mask_L$	PC_R	$Mask_R$	RPC
G	1111	-	-	-
B	1000	C	0111	G

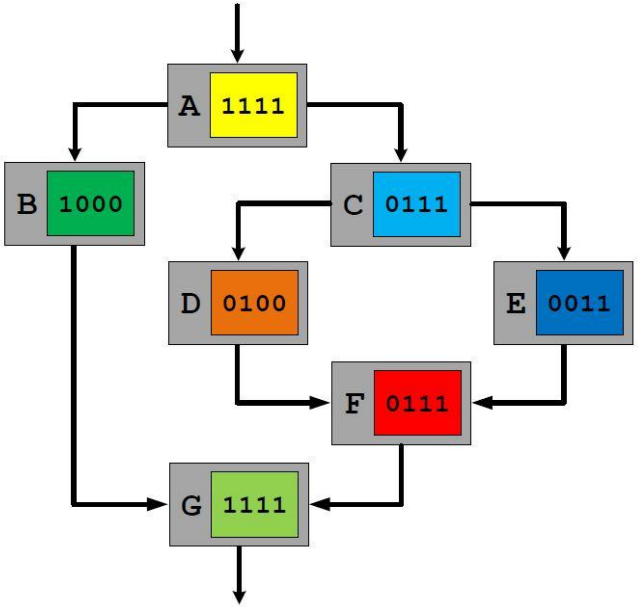
Dual-Path execution



Dual-path stack

	PC_L	$Mask_L$	PC_R	$Mask_R$	RPC
	G	1111	-	-	-
	B	1111	F	0111	G
TOS →	D	0100	E	0011	F

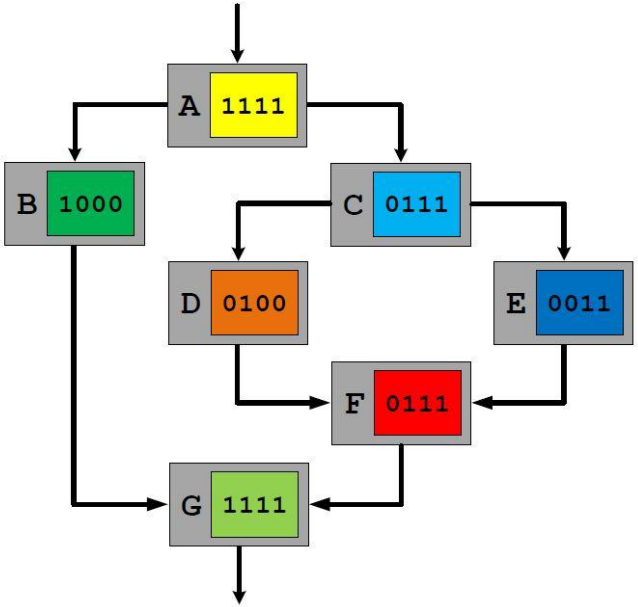
Dual-Path execution



Dual-path stack

	PC_L	$Mask_L$	PC_R	$Mask_R$	RPC
	G	1111	-	-	-
	B	1000	F	0111	G
TOS →	-	-	E	0011	F

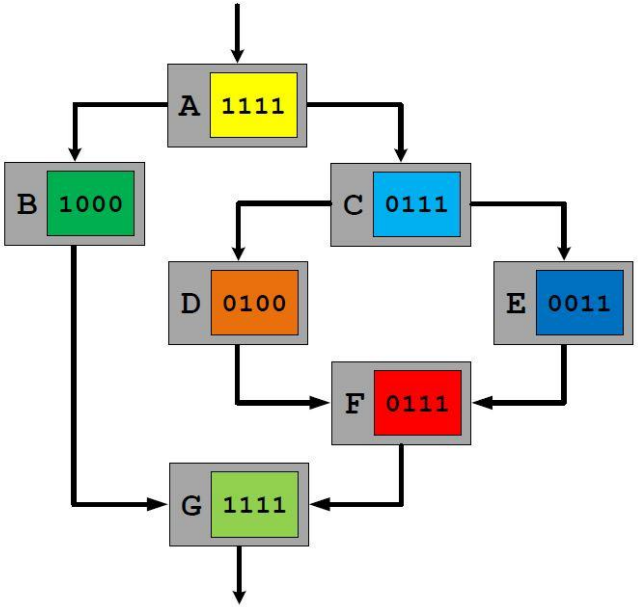
Dual-Path execution



Dual-path stack

	PC_L	$Mask_L$	PC_R	$Mask_R$	RPC
	G	1111	-	-	-
TOS →	B	1000	F	0111	G

Dual-Path execution



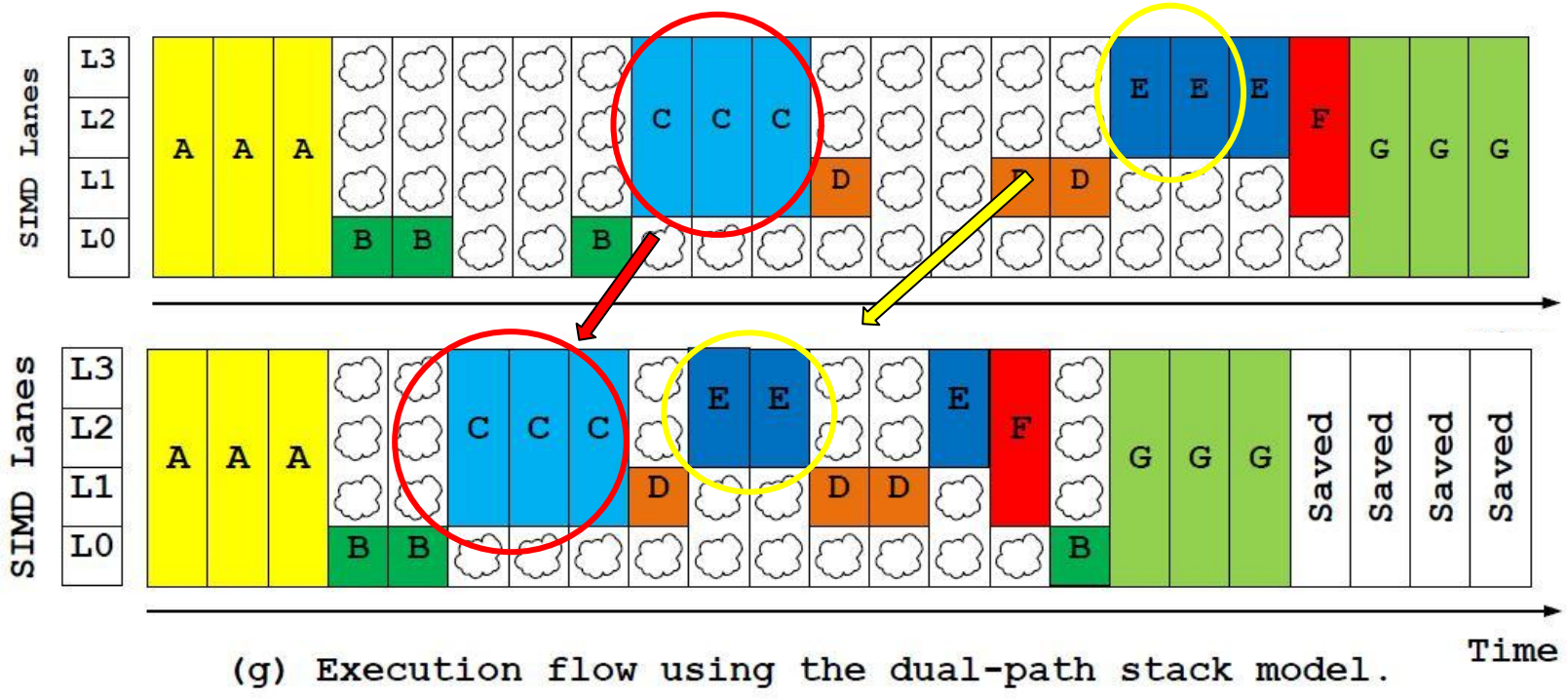
TOS →

Dual-path stack

PC_L	$Mask_L$	PC_R	$Mask_R$	RPC
G	1111	-	-	-

Dual-Path execution

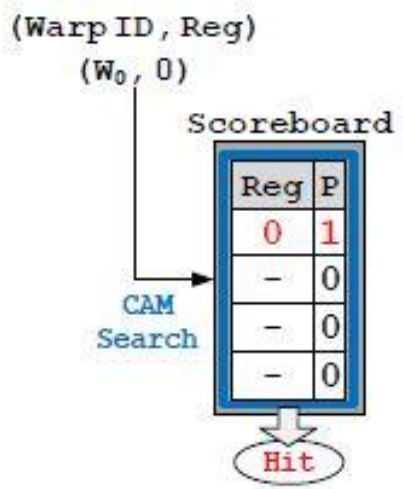
Compared with baseline architecture



(g) Execution flow using the dual-path stack model.

Scoreboard

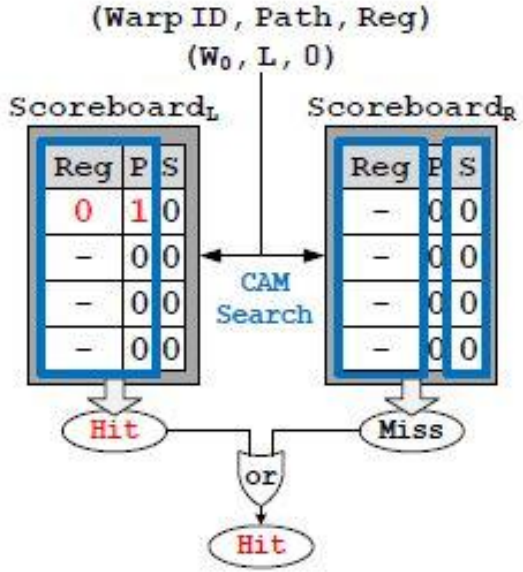
- Per-warp scoreboard to track data dependencies.
- Content-addressable-memory (CAM) structure: indexed with a register number and a warp ID which returns whether that register is pending write-back for that warp
- Once an instruction is scheduled for execution, the scoreboard is updated to show the instruction's destination register as pending.
- The pending P bit set for a register indicates that register has a pending write and all other registers dependent on that register must stall
- When the register is written back, the scoreboard is updated and the pending bit is cleared.
- A cleared P bit indicates the registers dependent on this register can proceed



(a) Input register number
compared in parallel with a
the scoreboard entrie
(Reg:P) field for a match.

Scoreboard

- In DPE, 2 divergent sub-warps can execute concurrently. To support concurrent paths per warp, the scoreboard scope is doubled to keep track of registers in both left and right paths separately.
- There exists a Shadow bit, S, in addition to the Pending P bit.
- P set indicates the register has a pending write
- P is copied to S when that register reaches a path divergence/ reconvergence
- While querying scoreboard, a register in a path checks the P in its own scoreboard or the S in the other path's scoreboard.
- If either is set, means the current path must stall



(b) Dependency is determined by OR-ing own path's (Reg:P) match and the other path's (Reg:S) match.

Scoreboard

Hit vs Miss ?

- Or-ring the scoreboards' outcomes for each path
- Hit if P in its own path or S in other's path is set
- Hit indicates path has data dependency and must stall to ensure correct execution when diverging and reconverging.
- Miss means path has no dependencies and can execute

Scoreboard

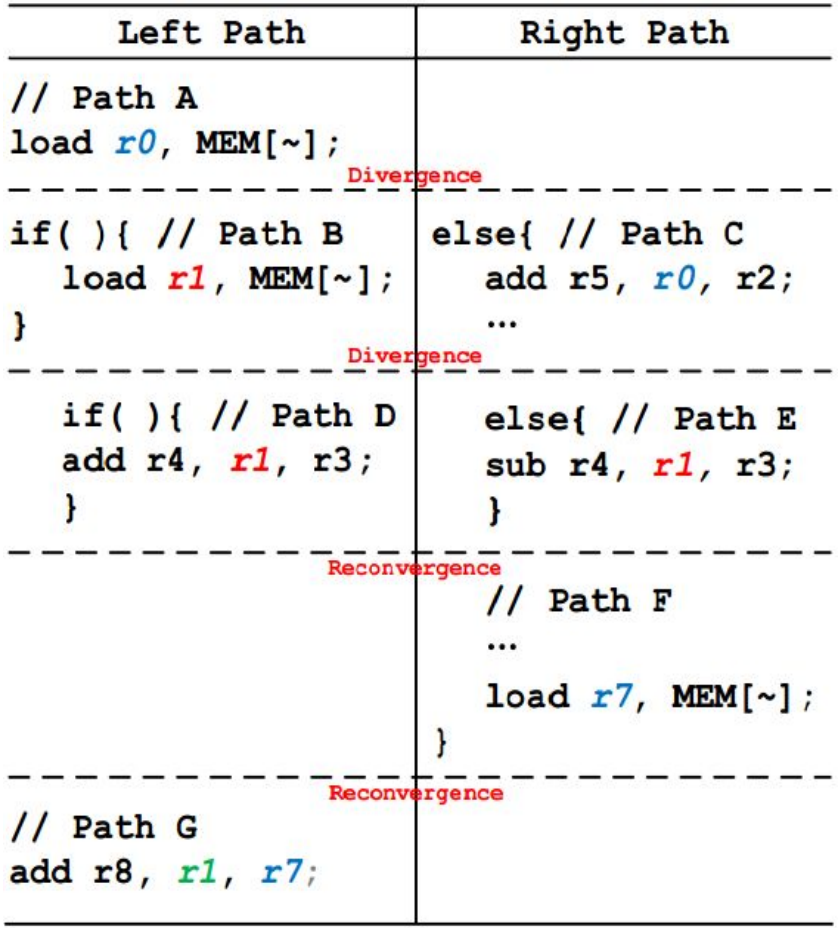
Scoreboard inserts stalls under the following scenarios:

1. Before/After Divergence
Path C reads r0, but must stall till r0 is written to by path A (true RAW dependency)

2. Before/After Reconvergence
Reading r7 on path G must stall till r7 is written on path F before reconvergence (true RAW dependency)

3. Registers with same register number but on different concurrent paths are unrelated but will be treated as false RAW dependency and insert stall

4. If the register number on two different paths is a destination in both paths concurrently, then writes to this register number from the two paths are actually unrelated but will be treated as a false WAW dependency. The score board will make the writes stall

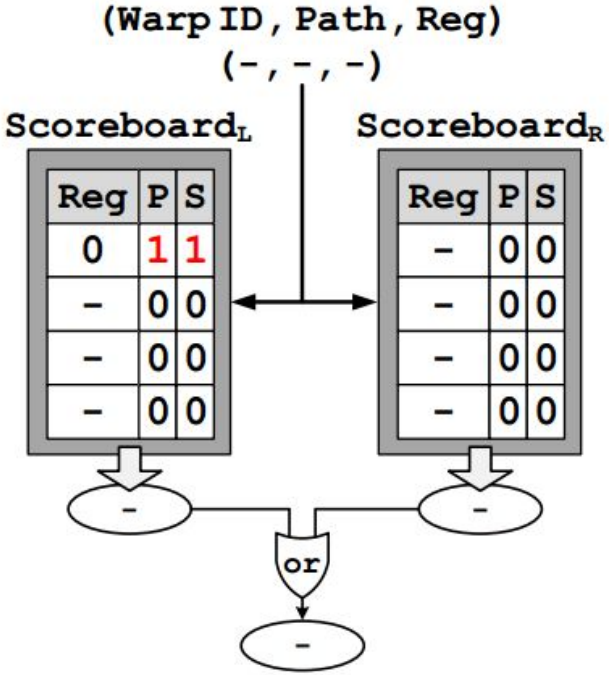


Scoreboard example

To illustrate how the scoreboard uses the P and S bits to check these dependencies across the 2 paths we have the following examples.

Initially, path A on the left path loads r0. Path A has a pending write and sets P.

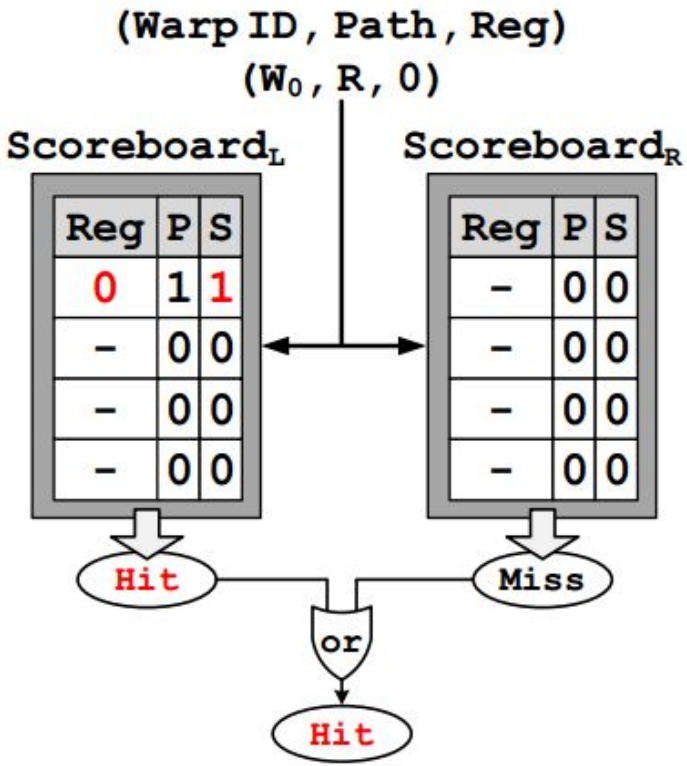
Later, when A reaches the BR(B-C) divergence, P is copied to S



Scoreboard example

When path C on the right path executes, it checks the S bit of the left path for r0. It finds S set which tells path C that path A has a pending write to r0 from pre-divergence.

Hence, C must wait/stall till A writes to r0.



Scoreboard example

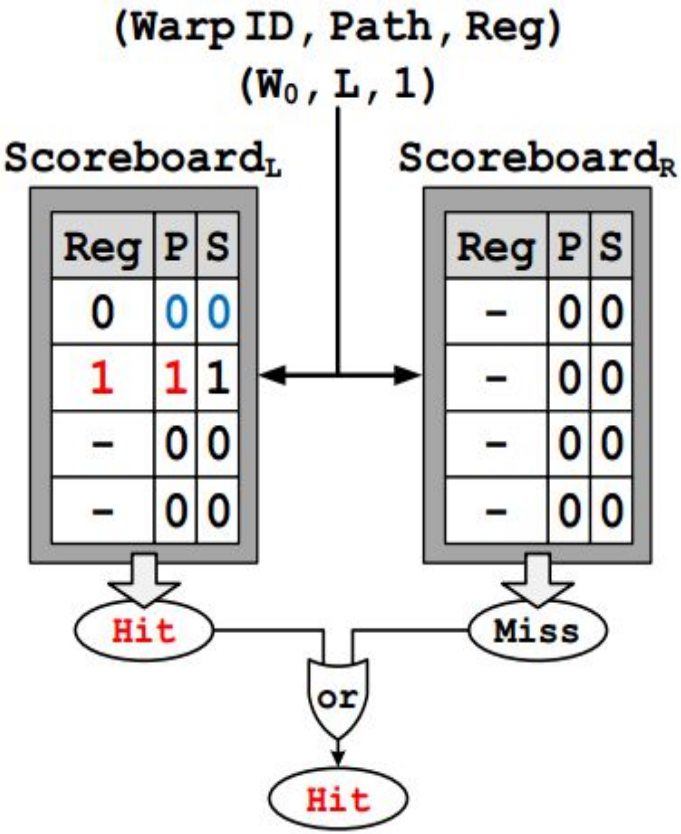
Once A is done loading r0, it clears its P and S bits. C can now proceed with its read of r0.

Next, path B on the left path is loading r1 and sets P on the left path to indicate a pending write to r1.

When B encounters BR(D-E) divergence, its P gets copied to S and S gets set.

Path D on the left path checks P on the same left path for r1 and stalls.

Path E on the right path checks S on the left path for r1 and stalls.



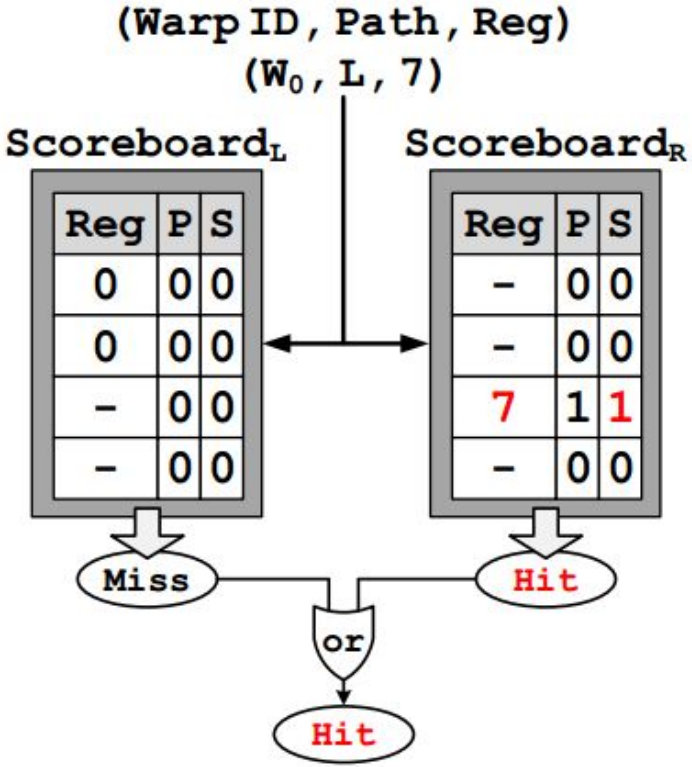
Scoreboard example

Path F on the right path is loading r7 and sets P.

When F reaches reconvergence, P is copied to S and S gets set.

Path G on left path checks S on the right path for r7 and finds it set, indicating a pending write. Hence, G stalls till S gets cleared.

This introduces a true RAW dependency.



Q. In Figure 7b the Pending bit is set for the register R1. Is it only cleared when all instructions (B , D and E who are changing R1) complete?

A. Pending bit is cleared when path B is done writing to r1. When B completes its write to r1, it clears both Pending and Shadow bits, indicating to other paths that its no longer having a pending write

Q. I don't think I fully understand what the scoreboard does. What does it mean to allow threads within the same warp to be issued Back-to-back?

A. The scoreboard is meant to keep track of true or false data dependencies between registers used in the left and right paths. The scoreboard is responsible for stalling dependent paths to ensure they get the correct values.

1 scoreboard structure for each warp. “Back to back” >> consecutive issue of threads in the warp. Because the left and right paths can actually execute simultaneously for the diverging sub-group of warps within a warp. Earlier, each sub-group executed in serial.

Warp Scheduler

- Schedules which ready warp to issue next
- Can have single scheduler or multiple parallel schedulers
- Nvidia’s Fermi GPU has 2 schedulers
  S0- Schedules even numbered warps
  S1- Schedules odd numbered warps
- DPE added to this further increases parallelism
- For a ready warp, there is a further right path and left path warp
- This doubles the number of ready warp entries competing to be issued

DPE and Scoreboard Benefits

Scoreboard

+ Conservative
- Introduces false dependencies
+ But is much simpler in design and operation
+ Much less hardware overhead and cost

- Non-conservative scoreboards are high cost, more hardware overhead
- Introduce only ~1% performance improvement over conservative ones

DPE

+ Increases parallelism
+ Permits atmost 2 divergent control flow paths to execute concurrently
+ requires only small changes to SPE model in terms of doubling the stack and scoreboard
+ Low cost
+ SIMD efficiency intact

Benchmarks

- 27 benchmarks
- 14 benchmarks shown here. Other 13 show identical results for DPE, DWS and SPE.
- Of the 14 benchmarks, only half of them benefit because of distinct left and right paths
- The other half do not result in distinct left and right paths that can be interleaved because many branches have only an if clause with no else.

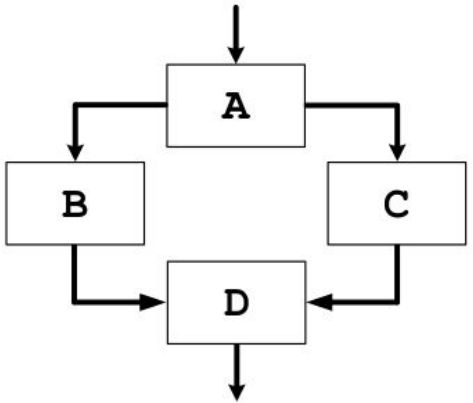
Interleavable		
Name	**Description**	**#Instr.**
LUD	LU Decomposition	39M
QSort	Quick Sort	60M
Stencil	3D Stencil Operation	115M
RAY	Ray Tracing	250M
LPS	Laplace Solver	72M
MUMpp	MUMmerGPU++	148M
MCML	Monte Carlo for ML Media	303B
Non-interleavable		
Name	**Description**	**#Instr.**
DXTC	DXT Compression	18B
BFS	Breadth-First Search	16M
PathFind	Path Finder	639M
NW	Needleman-Wunsch	51M
HOTSPOT	Hot-Spot	110M
BFS2	Breadth-First Search 2	26M
BACKP	Back Propagation	190M

6.1 Interleavable branches

< Code snippet from the kernel of ***LUD*** benchmark >

```
// Block A
if(threadIdx.x < BLOCK_SIZE){    // BR_B-C
    // Block B
    idx = threadIdx.x;
    array_offset = offset*matrix_dim+offset;
    for (i=0; i < BLOCK_SIZE/2; i++){ … }
    …
}
else{
    // Block C
    idx = threadIdx.x-BLOCK_SIZE;
    array_offset =(offset+BLOCK_SIZE/2)*matrix_dim+offset;
    for (i=BLOCK_SIZE/2; i < BLOCK_SIZE; i++){ … }
    …
}
// Block D
```

< Corresponding control flow graph >



6.1 Non-interleavable branches

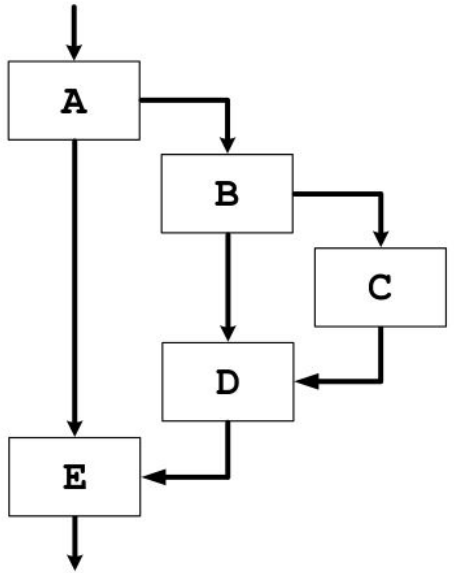
< Code snippet from the kernel of ***BFS*** benchmark >

```
int tid = blockIdx.x*MAX_THREADS_PER_BLOCK + threadIdx.x;

// Block A
if( tid<no_of_nodes && g_graph_mask[tid] )    // BR_B-E
{
    // Block B
    …

    // End of Path B
    if(!g_graph_visited[id])  // BR_C-D
    {
    // Block C
        …
    }
    // Block D
}
// Block E
```

< Corresponding control flow graph >



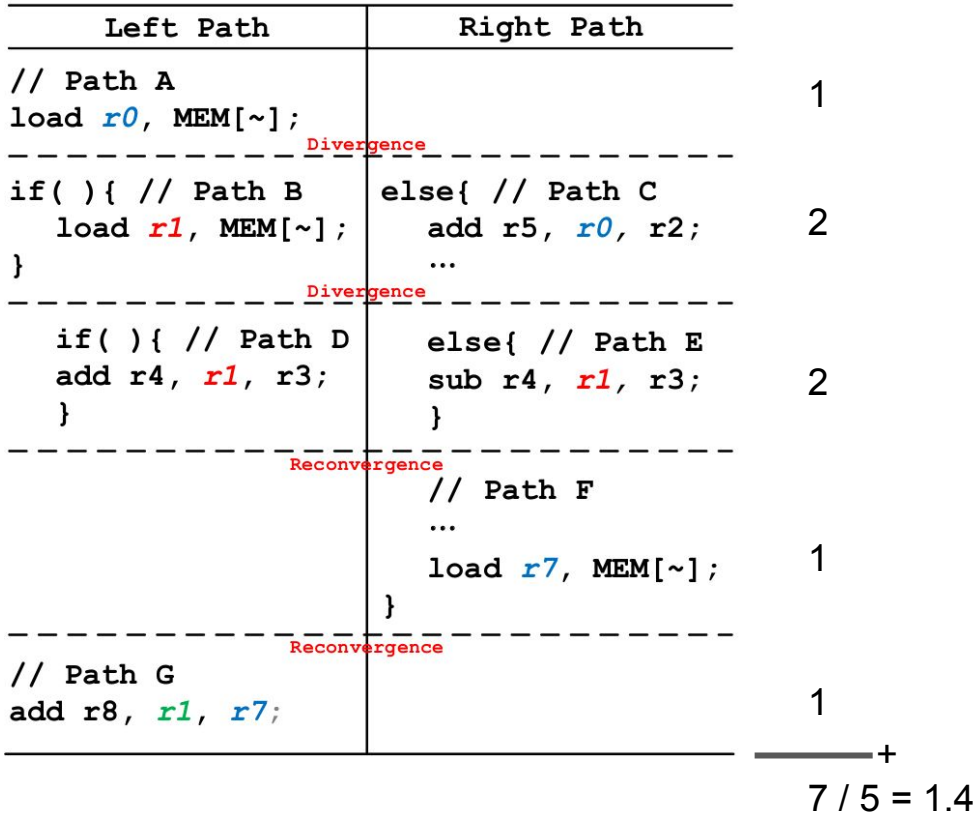
6.1 Interleavable vs non-interleavable

Interleavable			
Name	**Description**	**#Instr.**	**Ref.**
LUD	LU Decomposition	39M	[7]
QSort	Quick Sort	60M	[25]
Stencil	3D Stencil Operation	115M	[16]
RAY	Ray Tracing	250M	[5]
LPS	Laplace Solver	72M	[5]
MUMpp	MUMmerGPU++	148M	[13]
MCML	Monte Carlo for ML Media	303B	[15]
Non-interleavable			
Name	**Description**	**#Instr.**	**Ref.**
DXTC	DXT Compression	18B	[23]
BFS	Breadth-First Search	16M	[5]
PathFind	Path Finder	639M	[7]
NW	Needleman-Wunsch	51M	[7]
HOTSPOT	Hot-Spot	110M	[7]
BFS2	Breadth-First Search 2	26M	[7]
BACKP	Back Propagation	190M	[7]

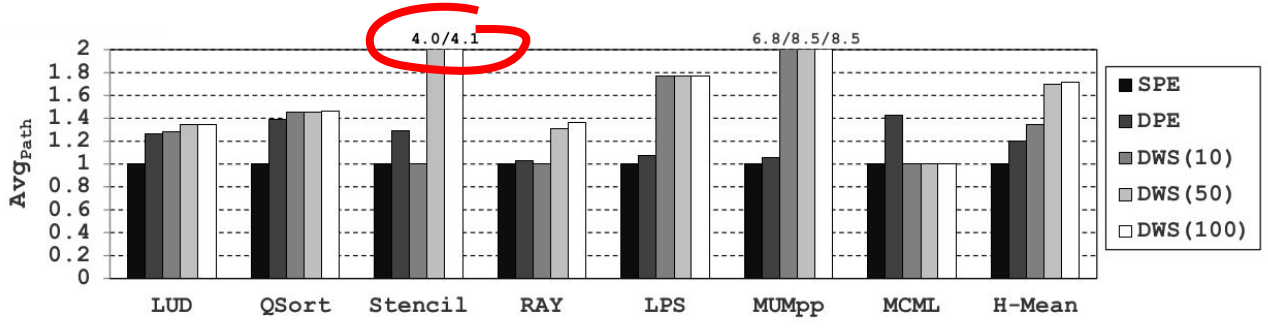
6.1 Potential for interleaving

$$Avg_{Path} = \frac{\sum_{i=1}^{N} NumPath_i}{N}$$

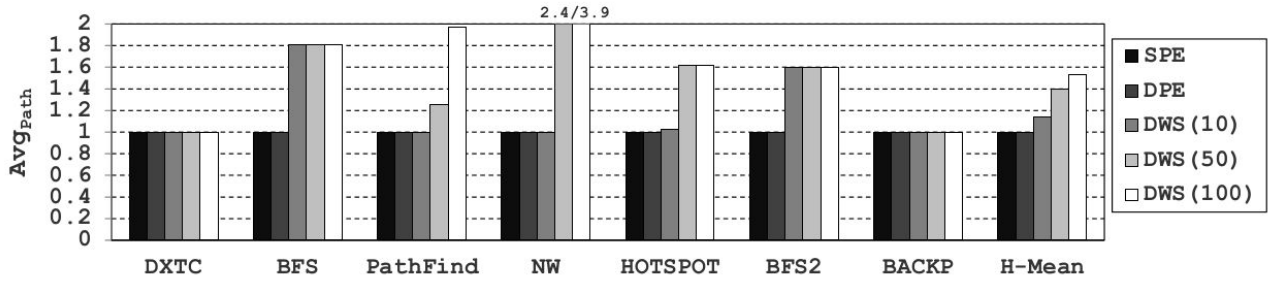
- SPE: Avg_{Path} = 1
- DWS: Avg_{Path} ≥ 1
- DPE
  - Interleavable: 1 < Avg_{Path} ≤ 2
  - Non-interleavable: Avg_{Path} = 1



6.1 Potential for interleaving



(a) Interleavable benchmarks.

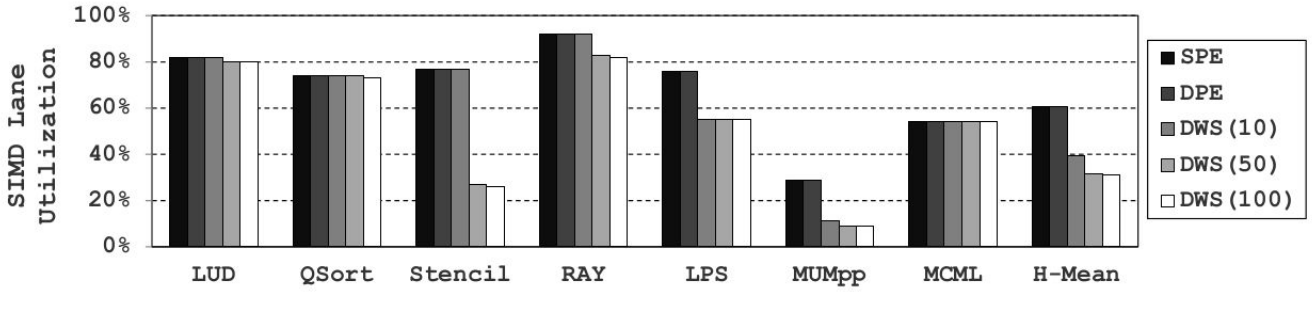


(b) Non-interleavable benchmarks.

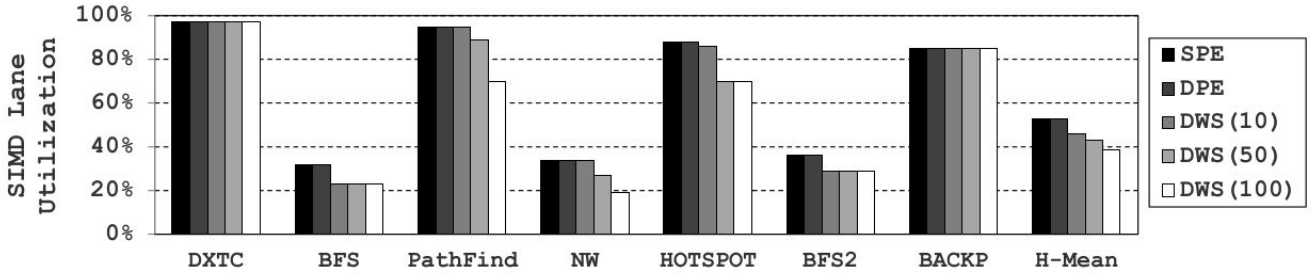
DPE: Avg_{Path} 20% higher on average than SPE for interleaved benchmarks.

DWS_{100}: Avg_{Path} 71% higher

6.1 SIMD lane utilization



(a) Interleavable benchmarks.

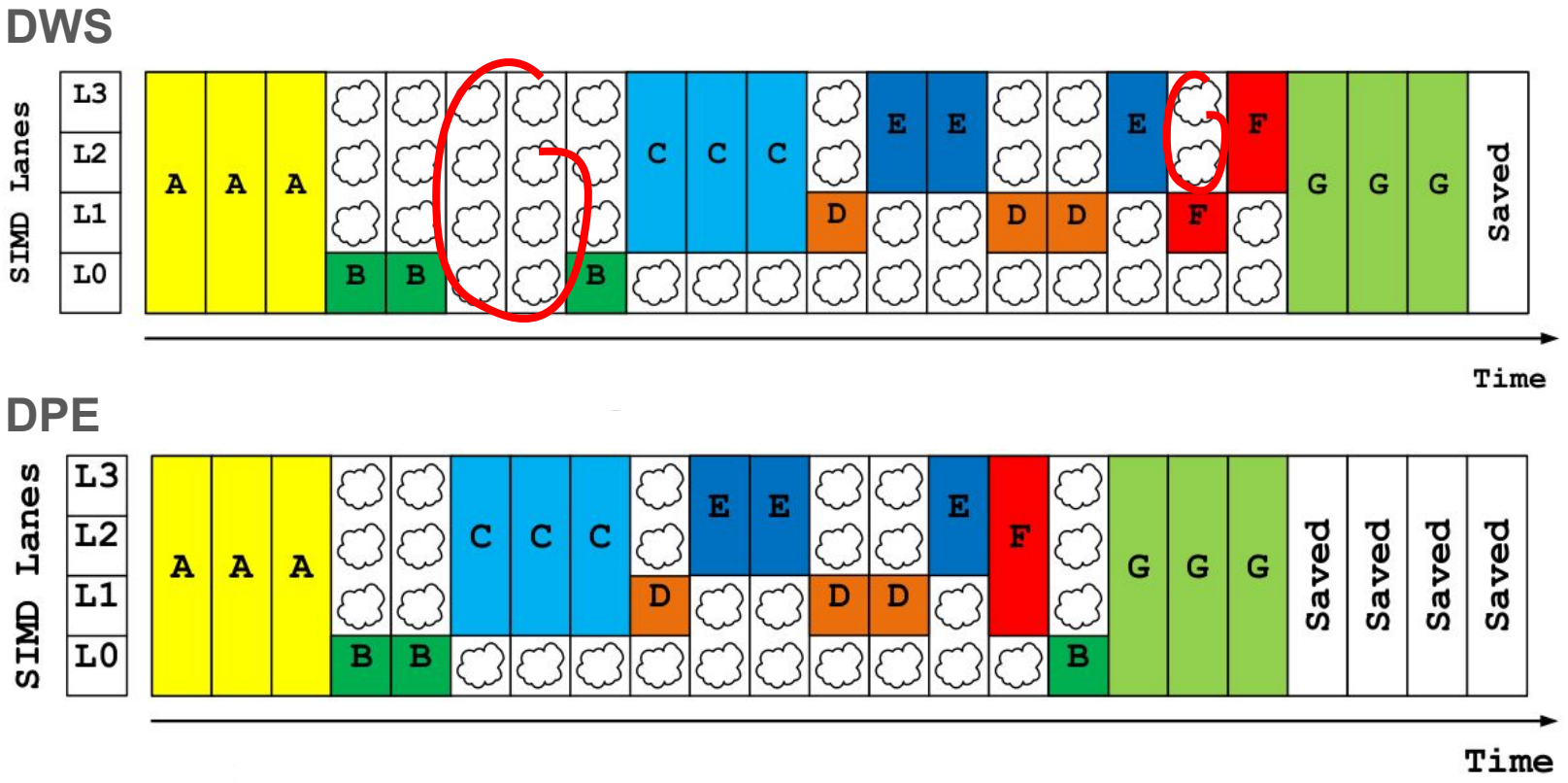


(b) Non-interleavable benchmarks.

DWS50/DWS100 reduce utilization by 48.1%/48.5% for interleavable and 18.6% and 27.1% for non-interleavable benchmarks

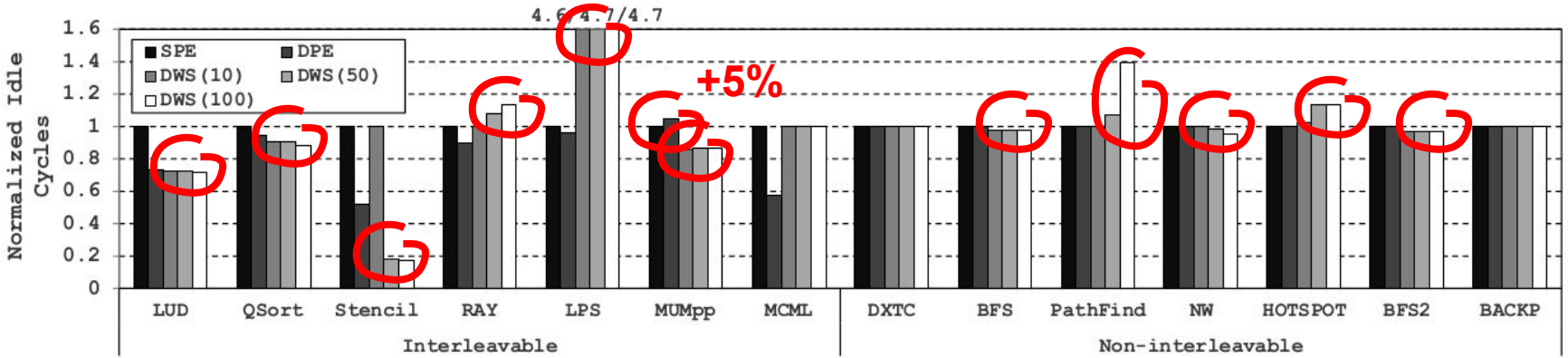
Due to overdivision.

6.1 SIMD lane utilization example

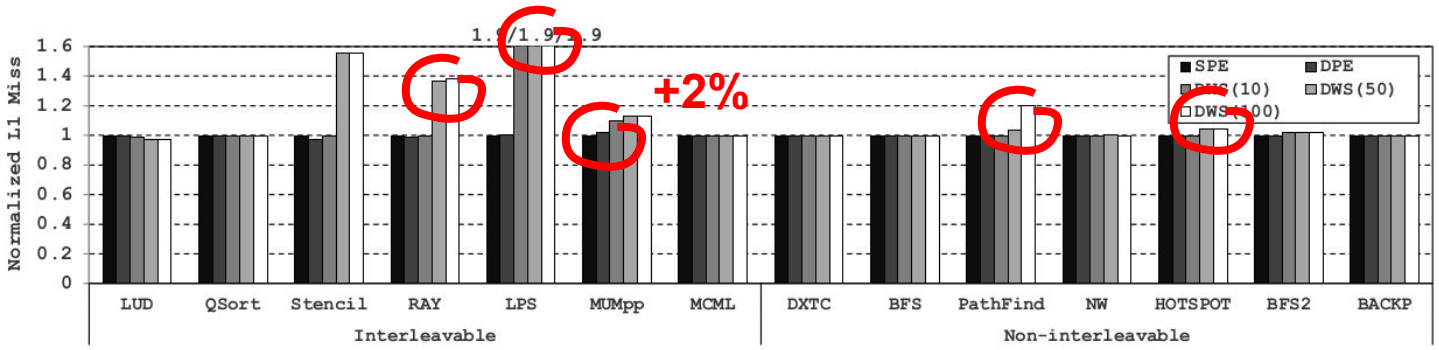


6.2 Idle cycles

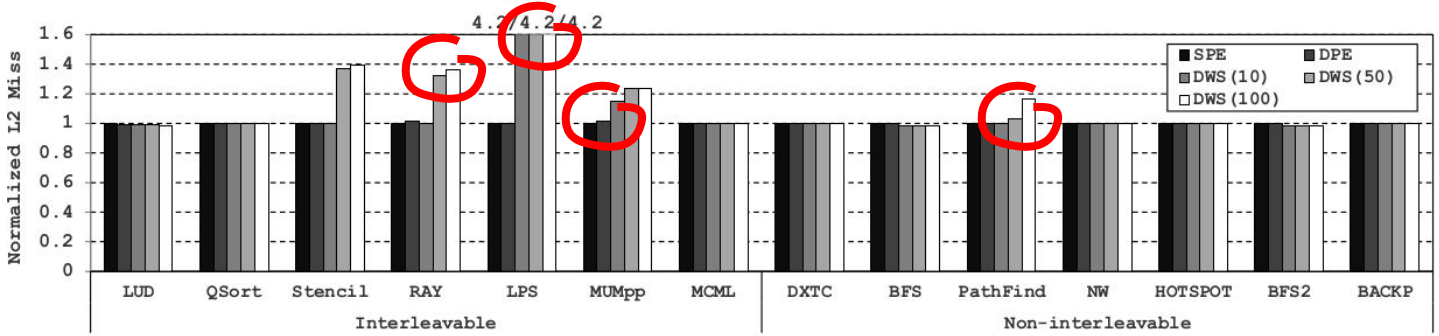
- DPE reduces idle cycles 19% on average for interleavable benchmarks.
- DWS can reduce idle cycles, but utilization decreases also.



6.2 Cache misses



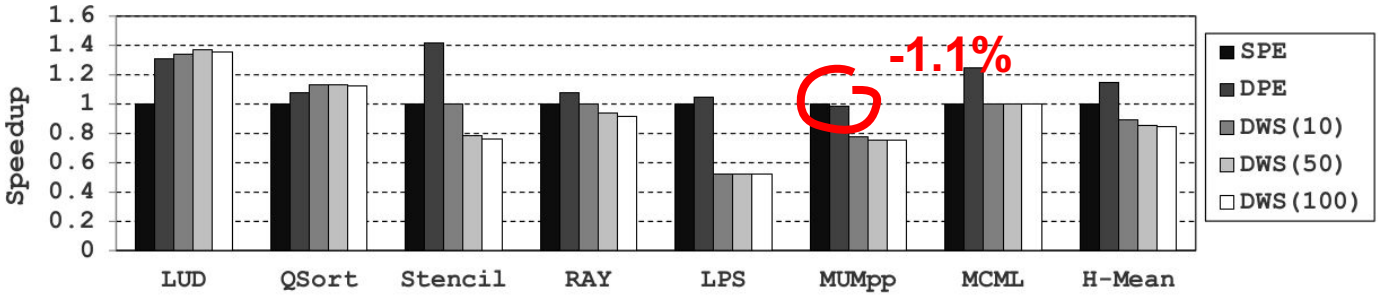
(b) Number of L1 misses.



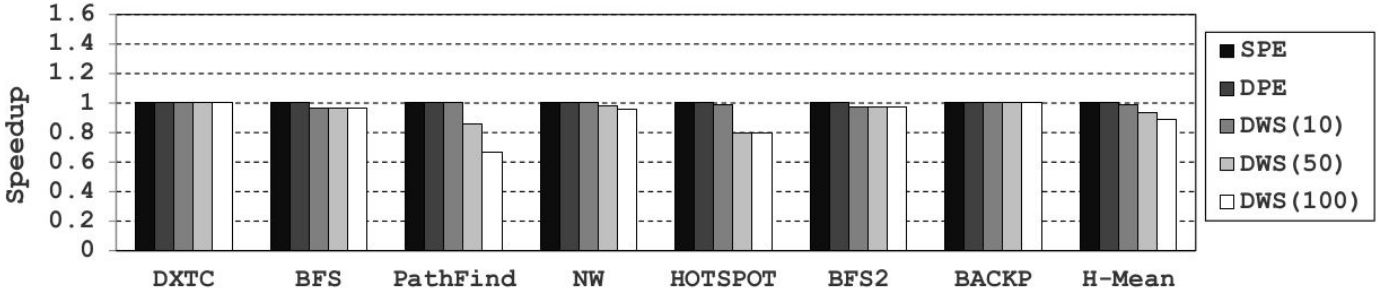
(c) Number of L2 misses.

Interleaving disrupts L1 cache access pattern.

6.3 Speedup



(a) Speedup over SPE among interleavable benchmarks.



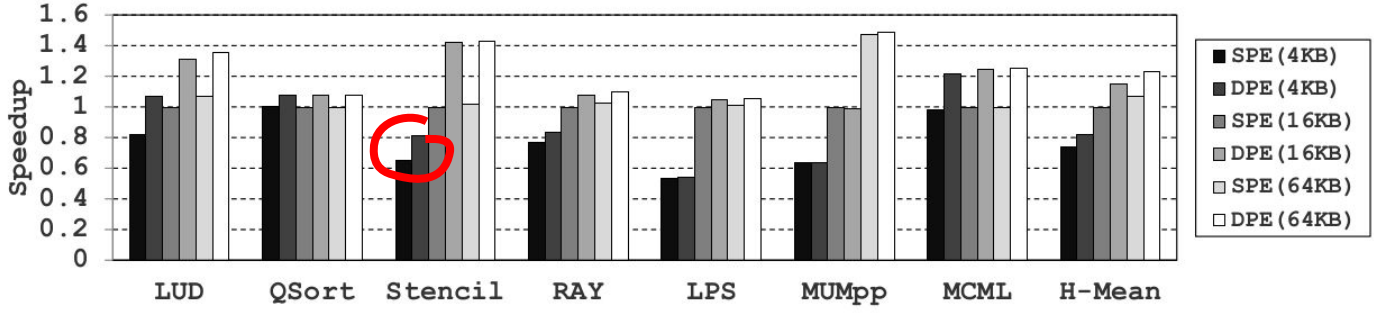
(b) Speedup over SPE among non-interleavable benchmarks.

DPE: 14.9% improvement for interleavable workloads.

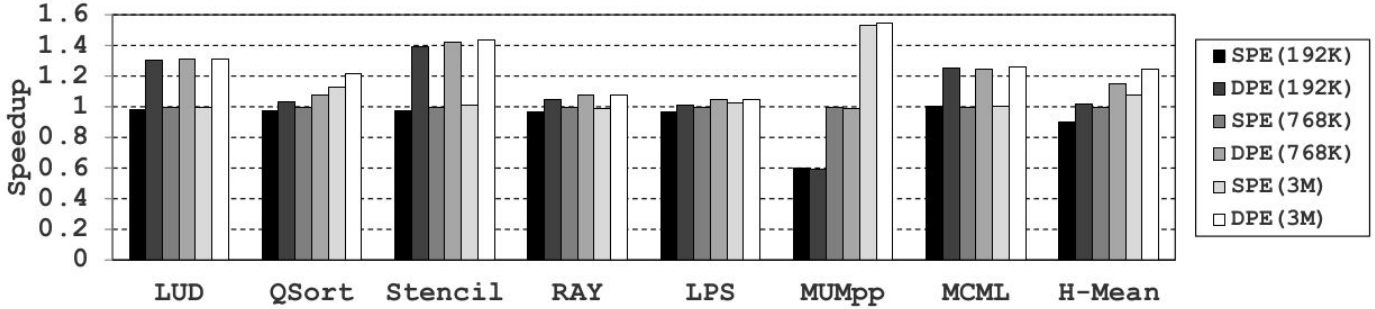
DWS performance varies.

Decrease of utilization outweighs TLP increase.

6.4 Sensitivity to cache size



(a) Performance sensitivity to different L1 cache size (normalized to SPE(16KB)).



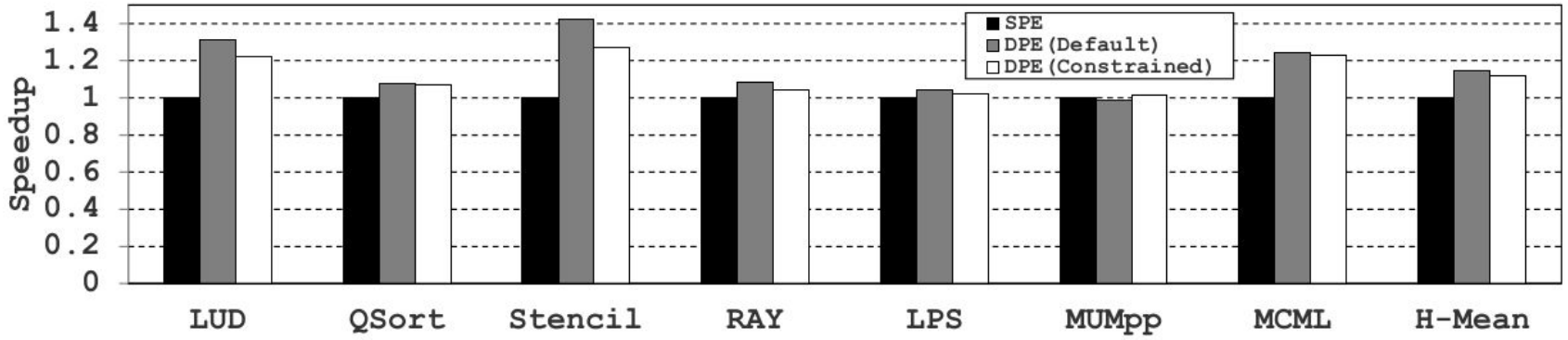
(b) Performance sensitivity to different L2 cache size (normalized to SPE(768K)).

Relative IPC improvement stable within ±4%/±2% for L1/L2.

Stencil: Absolute idle cycles improvement same, but relative differs.

6.4 Sensitivity to warp scheduler

- More aggressive scoreboard increased speedup by 1% (not shown).
- Constrained DPE: Path is only alternated on long-latency instruction.
  - Reduces speedup from 14.9% to 11.7% on average.



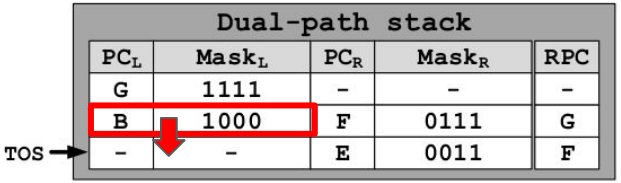
(c) Performance sensitivity when warp scheduler has limited context resources (normalized to SPE).

6.5 Implementation overhead

- Dual-path stack has negligible overhead w.r.t. single-path stack.
  - DPE needs longer entries (160-bit vs 96-bit).
  - Fewer entries needed for DPE (maximum observed 11 for SPE vs 7 for DPE).
- Addition of shadow bits to scoreboard adds 7-14% to scoreboard storage.
- Doubling number of scoreboards doubles scoreboard power and area.
- Warp scheduler doubles in size because instructions from both branches are stored.

7 Discussion

- Path forwarding: Shift branch up in stack to fill up entry of branch that finished.
  - < 2% Performance improvement for interleavable benchmarks.



- DPE for memory divergence
  - Limited benefit expected w.r.t. DWS.
- DPE with a software-managed reconvergence stack
  - Maintain PC and mask in hardware, and RPC in software.
  - A pop instruction informs hardware that a path has ended.

So just like the DWS paper, the two branches are not actually running in parallel, we are simply interleaving the threads?

Yes

Does this mean the only advantage comes from stalls when there is no active warps to run?

The SIMD utilization during non-idle cycles is also higher.

It seems like only the most immediate branch divergence paths can run in parallel. Is this true?

"Most immediate branch divergence path" is a bit vague. You probably mean "most recent". In the example, B and F could run in parallel. B diverged a lot earlier than F, so this is not the case.

Could you explain the relationship between lane utilization and the number of idle cycles?

Assuming this is about DWS, DWS reduces idle cycles, but lane utilization is reduced too. That is because idle cycles are filled with warp subdivision.

I don't understand why the relative performance differs a lot in different models.

DWS splits more warps than necessary. Split warps take multiple cycles as opposed to one cycle.

In Section 6.2, the third paragraph talks about counter intuitive results seen in RAY, LPS, PathFind and HOTSPOT with the statement "many interleaved warp-splits present a memory access pattern that performs poorly with the cache hierarchy". Could you explain this observation?

When you access data using a regular access pattern, a cache can take advantage of it by prefetching some data. Interleaved instructions may ruin the access pattern.

They briefly touch upon DPE for memory divergence. Does it actually seem like a feasible scheme to handle memory divergence at all? Considering that the parallelism is restricted to the right and left paths, if one path is hits and the other is misses how can they even be executed in parallel?

As before, we would not literally be executing paths in parallel, but we would interleave them.

Why not a quad-path execution model? Or 8 paths, or ...?

Because if-statements have only 2 branches… :-) Anyway, it is a tradeoff between area and performance. You could also use the area for more streaming multiprocessors for example.